A generalized Sylvester identity and fraction-free random Gaussian elimination

Report

Author(s):
Mulders, Thom

Publication date:
1997

Permanent link:
https://doi.org/10.3929/ethz-a-006652164

Originally published in:
Technical Report / ETH Zurich, Department of Computer Science 272

Technical Report No. 272, Department of Computer Science, ETH Zurich
August 1997

A GENERALIZED SYLVESTER IDENTITY AND FRACTION-FREE RANDOM GAUSSIAN ELIMINATION

THOM MULDERS
INSTITUTE OF SCIENTIFIC COMPUTING
ETH ZURICH, SWITZERLAND
MULDERS@INF.ETHZ.CH

Abstract. Sylvester's identity is a well-known identity which can be used to prove that certain Gaussian elimination algorithms are fraction-free. In this paper we will generalize Sylvester's identity and use it to prove that certain random Gaussian elimination algorithms are fraction-free. This can be used to yield fraction-free algorithms for solving $Ax = b$ $(x \geq 0)$ and for the simplex method in linear programming.

1. Introduction

Sylvester's identity is a well-known identity relating a hyperdeterminant of a matrix (i.e. a determinant of minors) to the determinant of that matrix.

Let R be a commutative ring and $A = (a_{ij})$ an $n \times m$ matrix over R. For $0 \leq k < \min(n, m)$, $k < i \leq n$ and $k < j \leq m$ define

$$a^{(k)}_{i,j} = \begin{vmatrix} a_{11} & \cdots & a_{1k} & a_{1j} \\ \vdots & & \vdots & \vdots \\ a_{k1} & \cdots & a_{kk} & a_{kj} \\ a_{i1} & \cdots & a_{ik} & a_{ij} \end{vmatrix}. \tag{1}$$

We can now state Sylvester's identity (for a proof see for example Bareiss (1968)).

Theorem 1. *(Sylvester's identity) If A is a square matrix of order n, then for $0 \leq k \leq n-1$ the following identity holds:*

$$|A| \left[a^{(k-1)}_{k,k}\right]^{n-k-1} = \begin{vmatrix} a^{(k)}_{k+1,k+1} & \cdots & a^{(k)}_{k+1,n} \\ \vdots & & \vdots \\ a^{(k)}_{n,k+1} & \cdots & a^{(k)}_{n,n} \end{vmatrix},$$

where $|A|$ denotes the determinant of A and $a^{(-1)}_{0,0} = 1$ by definition.

In Bareiss (1968) and Bareiss (1972) Sylvester's identity is used to prove that certain Gaussian elimination algorithms, used to transform a matrix to upper-triangular form, are fraction-free. In Bareiss (1968) these algorithms are extended in order to transform a matrix to diagonal form. Comparing the result of this diagonalization with Cramer's rule one can see that also these extended algorithms are fraction-free.

In this paper we will generalize Sylvester's identity. Using this generalized identity we can prove in a uniform way that above-mentioned algorithms and certain random Gaussian elimination algorithms (explained below) are fraction-free.

The random Gaussian elimination algorithms can be used for solving $Ax = b$ $(x \geq 0)$ and in the simplex method for solving linear programs. In this way we get fraction-free algorithms for these applications.

2. A generalized Sylvester identity

In order to have a nice framework in which we can state and prove our results we first introduce some notations.

Let R and A be as in the previous section. For $1 \leq i_1, \ldots, i_k \leq n$ and $1 \leq j_1, \ldots, j_k \leq m$ define the matrix

$$[(i_1, \ldots, i_k), (j_1, \ldots, j_k)]_A = \begin{pmatrix} a_{i_1 j_1} & \cdots & a_{i_1 j_k} \\ \vdots & & \vdots \\ a_{i_k j_1} & \cdots & a_{i_k j_k} \end{pmatrix}.$$

Let $S = \{((i_1, j_1), \ldots, (i_k, j_k)) \mid 1 \leq i_s, j_s$ for all s and $i_1, \ldots, i_k$ all different$\}$ and define the equivalence $\sim$ on S by

$$\begin{aligned} &((i_1, j_1), \ldots, (i_k, j_k)) \sim ((u_1, v_1), \ldots, (u_r, v_r)) \Leftrightarrow \\ &\quad k = r \text{ and there is a permutation } \alpha \text{ of } \{1, \ldots, k\} \text{ such that} \\ &\quad i_{\alpha(s)} = u_s \text{ and } j_{\alpha(s)} = v_s \text{ for all } s. \end{aligned}$$

The equivalence class of $((i_1, j_1), \ldots, (i_k, j_k))$ in $S/\sim$ is denoted by $[(i_1, j_1), \ldots, (i_k, j_k)]$. Notice that we allow $k = 0$.

For $1 \leq i_1, \ldots, i_k \leq n$ all different, $1 \leq j_1, \ldots, j_k \leq m$ we define the determinant

$$a^{[(i_1, j_1), \ldots, (i_k, j_k)]} = |\,[(i_1, \ldots, i_k), (j_1, \ldots, j_k)]_A\,| \qquad (a^{[]} = 1).$$

It is easy to see that this is well-defined.

We define an operation $\leftarrow$ on $S/\sim$ by:

$$\begin{aligned} &[(i_1, j_1), \ldots, (i_k, j_k)] \leftarrow [(u_1, v_1), \ldots, (u_r, v_r)] = \\ &\quad [\overline{(i_1, j_1)}, \ldots, \overline{(i_k, j_k)}, (u_1, v_1), \ldots, (u_r, v_r)], \end{aligned}$$

where

$$\overline{(i_s, j_s)} = \begin{cases} (i_s, j_s) & \text{if } i_s \notin \{u_1, \ldots, u_r\} \\ \text{not present} & \text{if } i_s \in \{u_1, \ldots, u_r\} \end{cases}$$

So $\leftarrow$ replaces (i_s, j_s) by (u_t, v_t) if $i_s = u_t$ and adds (u_t, v_t) if $u_t \neq i_s$ for all s. Notice that $\leftarrow$ is associative, i.e. for all $s_1, s_2, s_3 \in S/\sim$ we have $(s_1 \leftarrow s_2) \leftarrow s_3 = s_1 \leftarrow (s_2 \leftarrow s_3)$. So we can leave out parentheses and write $s_1 \leftarrow s_2 \leftarrow s_3$ without ambiguity. Also notice that $\leftarrow$ is not commutative. We have however that $[(i_1, j_1), \ldots, (i_k, j_k)] \leftarrow [(u_1, v_1), \ldots, (u_r, v_r)] = [(u_1, v_1), \ldots, (u_r, v_r)] \leftarrow [(i_1, j_1), \ldots, (i_k, j_k)]$ if $\{i_1, \ldots, i_k\}$ and $\{u_1, \ldots, u_r\}$ are disjoint.

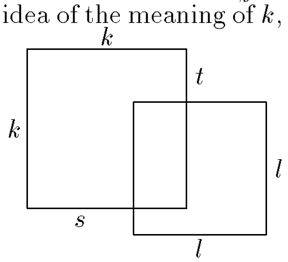
For $1 \leq i_1, \ldots, i_k \leq n$ all different, $1 \leq j_1, \ldots, j_k \leq m$, $1 \leq i \leq n$ and $1 \leq j \leq m$ we define the determinant

$$a_{i,j}^{[(i_1, j_1), \ldots, (i_k, j_k)]} = a^{[(i_1, j_1), \ldots, (i_k, j_k)] \leftarrow [(i,j)]}. \tag{2}$$

So when $i \neq i_s$ for all s, the matrix $[(i_1, \ldots, i_k), (j_1, \ldots, j_k)]_A$ is extended with the ith row and jth column of A before taking the determinant. When $i = i_s$, the sth column of $[(i_1, \ldots, i_k), (j_1, \ldots, j_k)]_A$ is replaced by the jth column of A before taking the determinant.

When $i_s = s$ and $j_s = s$ for all s, we also write $a_{i,j}^{(k)}$ instead of $a_{i,j}^{[(i_1, j_1), \ldots, (i_k, j_k)]}$. Notice that when $i, j > k$ then this definition of $a_{i,j}^{(k)}$ coincides with (1) (so (1) is a special case of (2)) but when $i < k$ then $a_{i,j}^{(k)}$ is not the same as $a_{ij}^{(k+1)}$ in Bareiss (1972) (definition (2.23)).

Now we can formulate the generalized Sylvester identity. It is a generalization since it allows in the hyperdeterminant entries $a_{i,j}^{(k)}$ where i and/or j are $\leq k$. The following picture will give an idea of the meaning of k, s, t and l in the next theorem.



Theorem 2. *(Generalized Sylvester identity) For $0 \leq k \leq \min(n,m)$, $0 \leq s,t \leq k$ and $1 \leq l \leq \min(n-t, m-s)$, the following identity holds:*

$$a^{[(1,1),\dots,(t,t),(t+1,s+1),\dots,(t+l,s+l)]}\left[a_{k,k}^{(k-1)}\right]^{l-1} = \begin{vmatrix} a_{t+1,s+1}^{(k)} & \cdots & a_{t+1,s+l}^{(k)} \\ \vdots & & \vdots \\ a_{t+l,s+1}^{(k)} & \cdots & a_{t+l,s+l}^{(k)} \end{vmatrix}.$$

Proof. Let $B = (b_{ij})$ be the following square matrix of order $k + l$:

$$B = \begin{pmatrix} a_{11} & \cdots & a_{1t} & a_{1\,t+1} & \cdots & a_{1k} & a_{1\,s+1} & \cdots & a_{1\,s+l} \\ \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ a_{t1} & \cdots & a_{tt} & a_{t\,t+1} & \cdots & a_{tk} & a_{t\,s+1} & \cdots & a_{t\,s+l} \\ a_{t+1\,1} & \cdots & a_{t+1\,t} & a_{t+1\;t+1} & \cdots & a_{t+1\,k} & a_{t+1\;s+1} & \cdots & a_{t+1\;s+l} \\ \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ a_{k1} & \cdots & a_{kt} & a_{k\,t+1} & \cdots & a_{kk} & a_{k\,s+1} & \cdots & a_{k\,s+l} \\ & & & -1 & & & & & \\ & & & & \ddots & & & & \\ & & & & & -1 & & & \\ a_{k+1\,1} & \cdots & a_{k+1\,t} & a_{k+1\;t+1} & \cdots & a_{k+1\,k} & a_{k+1\;s+1} & \cdots & a_{k+1\;s+l} \\ \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ a_{t+l\,1} & \cdots & a_{t+l\,t} & a_{t+l\;t+1} & \cdots & a_{t+l\,k} & a_{t+l\;s+1} & \cdots & a_{t+l\;s+l} \end{pmatrix}.$$

Applying Sylvester's identity on B yields:

$$|B|\left[b_{k,k}^{(k-1)}\right]^{l-1} = \begin{vmatrix} b_{k+1,k+1}^{(k)} & \cdots & b_{k+1,k+l}^{(k)} \\ \vdots & & \vdots \\ b_{k+l,k+1}^{(k)} & \cdots & b_{k+l,k+l}^{(k)} \end{vmatrix}.$$

Now it is easy to see that $|B| = a^{[(1,1),\dots,(t,t),(t+1,s+1),\dots,(t+l,s+l)]}$, $b_{k,k}^{(k-1)} = a_{k,k}^{(k-1)}$ and $b_{k+i,k+j}^{(k)} = a_{t+i,s+j}^{(k)}$ for $k - t < i \leq l$ and $1 \leq j \leq l$. From the identity

$$a_{x,y}^{(k)} = \begin{vmatrix} a_{11} & \cdots & \cdots & a_{1x} & \cdots & \cdots & a_{1k} & a_{1y} \\ \vdots & & & \vdots & & \cdots & \vdots & \vdots \\ a_{k1} & \cdots & \cdots & a_{kx} & \cdots & \cdots & a_{kk} & a_{ky} \\ 0 & \cdots & 0 & -1 & 0 & \cdots & 0 & 0 \end{vmatrix} \qquad (1 \leq x \leq k)$$

it follows easily that also $b^{(k)}_{k+i,k+j} = a^{(k)}_{t+i,s+j}$ for $1 \leq i \leq k-t$ and $1 \leq j \leq l$. This proves the theorem. $\square$

Notice that for $m = n$, $t = s = k$ and $l = n - k$ this is exactly Sylvester's identity.

3. Random Gaussian elimination

When performing a Gaussian elimination algorithm, in order to transform a matrix to diagonal form, one chooses a pivot (or an $r \times r$ pivot region when one performs an r-step elimination algorithm) which is used to make the entries in a column (or r columns) equal to 0 (this is called pivoting). These pivots are usually chosen in the columns and rows which did not yet contribute to a pivot. In the easiest case, for $r = 1$, the first pivot is chosen in the first column and first row, the second in the second column and second row, and so on.

Now in some applications (e.g. in solving $Ax = b$ $(x \geq 0)$ or in the simplex method in linear programming) it may be that one has to choose a pivot in a column and/or row which already contributed to an earlier pivot. When doing so we will call this random Gaussian elimination. The following example will illustrate the difference.

Example We will transform the matrix

$$A = \begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix}$$

to diagonal form (up to a permutation in the second computation). In the first computation we choose the pivots in different columns and rows, in the second we use the first column and row twice. The pivots are marked by a box.

$$\begin{pmatrix} \boxed{1} & 1 \\ 2 & 3 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 1 \\ 0 & \boxed{1} \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} \boxed{1} & 1 \\ 2 & 3 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & \boxed{1} \\ 0 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 1 \\ \boxed{-1} & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

In the sequel we will give an algorithm which will perform random Gaussian elimination and, using the generalized Sylvester identity, we can show that this algorithm is fraction-free. For this we need the following lemma.

Lemma 1. *Let A be an $n \times m$ matrix. Let $i_1, \ldots, i_k \in \{1, \ldots, n\}$ be different, $j_1, \ldots, j_k \in \{1, \ldots, m\}$ and $B = (b_{xy})$ be such that $b_{xy} = a^{[(i_1,j_1),\ldots,(i_k,j_k)]}_{x,y}$. Let $u_1, \ldots, u_r \in \{1, \ldots, n\}$ be different and $v_1, \ldots, v_r \in \{1, \ldots, m\}$. Then for $i \in \{1, \ldots, n\} \setminus \{u_1, \ldots, u_r\}$ and $j \in \{1, \ldots, m\}$ we have the following identity:*

$$b^{[(u_1,v_1),\ldots,(u_r,v_r)]}_{i,j} = \left[a^{[(i_1,j_1),\ldots,(i_k,j_k)]}\right]^r a^{[(i_1,j_1),\ldots,(i_k,j_k)]\leftarrow[(u_1,v_1),\ldots,(u_r,v_r)]}_{i,j}.$$

Proof. We distinguish two cases:

1. $i \notin \{i_1, \ldots, i_k\}$: We may assume that $i_1, \ldots, i_t \notin \{u_1, \ldots, u_r\}$, $i_{t+1} = u_1, \ldots, i_k = u_{k-t}$ and $u_{k-t+1}, \ldots, u_r \notin \{i_1, \ldots, i_k\}$. Let

$$C = [(i_1, \ldots, i_k, u_{k-t+1}, \ldots, u_r, i), (j_1, \ldots, j_k, v_1, \ldots, v_r, j)]_A \, .$$

Then

$$\begin{aligned} b_{u_x v_y} &= a^{[(i_1,j_1),\dots,(i_k,j_k)]}_{u_x,v_y} \\ &= c^{(k)}_{t+x,k+y}. \end{aligned}$$

In the same way we find $b_{i\,v_y} = c^{(k)}_{t+r+1,k+y}$, $b_{u_x\,j} = c^{(k)}_{t+x,k+r+1}$ and $b_{i\,j} = c^{(k)}_{t+r+1,k+r+1}$. From this we see that

$$\begin{aligned} & b^{[(u_1,v_1),\dots,(u_r,v_r)]}_{i,j} = \\ &= b^{[(u_1,v_1),\dots,(u_r,v_r),(i,j)]} \\ &= \begin{vmatrix} c^{(k)}_{t+1,k+1} & \cdots & c^{(k)}_{t+1,k+r} & c^{(k)}_{t+1,k+r+1} \\ \vdots & & \vdots & \vdots \\ c^{(k)}_{t+r,k+1} & \cdots & c^{(k)}_{t+r,k+r} & c^{(k)}_{t+r,k+r+1} \\ c^{(k)}_{t+r+1,k+1} & \cdots & c^{(k)}_{t+r+1,k+r} & c^{(k)}_{t+r+1,k+r+1} \end{vmatrix}. \end{aligned}$$

Using the generalized Sylvester identity on C with $s = k$ and $l = r+1$ we see that this last determinant equals

$$\begin{aligned} & \left[c^{(k-1)}_{k,k}\right]^r c^{[(1,1),\dots,(t,t),(t+1,k+1),\dots,(t+r+1,k+r+1)]} = \\ &= \left[a^{[(i_1,j_1),\dots,(i_k,j_k)]}\right]^r a^{[(i_1,j_1),\dots,(i_t,j_t),(u_1,v_1),\dots,(u_r,v_r),(i,j)]} \\ &= \left[a^{[(i_1,j_1),\dots,(i_k,j_k)]}\right]^r a^{[(i_1,j_1),\dots,(i_t,j_t),(u_1,v_1),\dots,(u_r,v_r)]}_{i,j}, \end{aligned}$$

which proves the lemma in this case.

2. $i \in \{i_1,\dots,i_k\}$: We may assume that $i_1,\dots,i_t \notin \{i,u_1,\dots,u_r\}$, $i_{t+1} = i, i_{t+2} = u_1, \dots, i_k = u_{k-t-1}$ and $u_{k-t},\dots,u_r \notin \{i_1,\dots,i_k\}$. Let

$$C = [(i_1,\dots,i_k,u_{k-t},\dots,u_r),(j_1,\dots,j_k,j,v_1,\dots,v_r)]_A.$$

Then

$$\begin{aligned} b_{u_x v_y} &= a^{[(i_1,j_1),\dots,(i_k,j_k)]}_{u_x,v_y} \\ &= c^{(k)}_{t+1+x,k+1+y}. \end{aligned}$$

In the same way we find $b_{i\,v_y} = c^{(k)}_{t+1,k+1+y}$, $b_{u_x\,j} = c^{(k)}_{t+1+x,k+1}$ and $b_{i\,j} = c^{(k)}_{t+1,k+1}$. From this we see that

$$\begin{aligned} & b^{[(u_1,v_1),\dots,(u_r,v_r)]}_{i,j} = \\ &= b^{[(i,j),(u_1,v_1),\dots,(u_r,v_r)]} \\ &= \begin{vmatrix} c^{(k)}_{t+1,k+1} & c^{(k)}_{t+1,k+2} & \cdots & c^{(k)}_{t+1,k+1+r} \\ c^{(k)}_{t+2,k+1} & c^{(k)}_{t+2,k+2} & \cdots & c^{(k)}_{t+2,k+1+r} \\ \vdots & \vdots & & \vdots \\ c^{(k)}_{t+1+r,k+1} & c^{(k)}_{t+1+r,k+2} & \cdots & c^{(k)}_{t+1+r,k+1+r} \end{vmatrix}. \end{aligned}$$

Using the generalized Sylvester identity on C with $s = k$ and $l = r + 1$ we see that this last determinant equals

$$\begin{aligned} \left[c_{k,k}^{(k-1)}\right]^r c^{[(1,1),\ldots,(t,t),(t+1,k+1),\ldots,(t+r+1,k+r+1)]} &= \\ = \left[a^{[(i_1,j_1),\ldots,(i_k,j_k)]}\right]^r a^{[(i_1,j_1),\ldots,(i_t,j_t),(i,j),(u_1,v_1),\ldots,(u_r,v_r)]} \\ = \left[a^{[(i_1,j_1),\ldots,(i_k,j_k)]}\right]^r a_{i,j}^{[(i_1,j_1),\ldots,(i_t,j_t),(i_{t+1},j_{t+1}),(u_1,v_1),\ldots,(u_r,v_r)]}, \end{aligned}$$

which proves the lemma in this case.

□

3.1. **The algorithm.** Now we can state the fraction-free random Gaussian elimination algorithm. In fact we will give a general scheme, representing a whole class of algorithms. From now on we assume that R is an integral domain.

Algorithm Fraction-free random Gaussian elimination
Description Fraction-free diagonalization of A
$B := \text{copy}(A)$;
while not satified **do**
  comment: Now there are $1 \le i_1, \ldots, i_k \le n$ all different, $1 \le j_1, \ldots, j_k \le m$ such that $B = (b_{xy})$ and $b_{xy} = a_{x,y}^{[(i_1,j_1),\ldots,(i_k,j_k)]}$
  Choose $1 \le u_1, \ldots, u_r \le n$ all different, $1 \le v_1, \ldots, v_r \le m$;
  for $x \in \{1, \ldots, n\} \setminus \{u_1, \ldots, u_r\}$ and $1 \le y \le m$ **do**
    Replace b_{xy} by $b_{x,y}^{[(u_1,v_1),\ldots,(u_r,v_r)]} / \left[a^{[(i_1,j_1),\ldots,(i_k,j_k)]}\right]^r$
  od;
  Diagonalize$(B, [(i_1, j_1), \ldots, (i_k, j_k)], ((u_1, v_1), \ldots, (u_r, v_r)))$
od;

So first rows $u_1, \ldots, u_r$ are used to clear columns $v_1, \ldots, v_r$ outside rows $u_1, \ldots, u_r$. Then rows $u_1, \ldots, u_r$ are diagonalized (in columns $v_1, \ldots, v_r$).

Later we will say something on the choice of $u_1, \ldots, u_r$ and $v_1, \ldots, v_r$. The stopping condition in the algorithm depends on the particular application. One can for example stop as soon as one has found an $n \times n$ diagonal submatrix, or one can impose some extra conditions, as is done in the simplex method.

Algorithm Diagonalize
Description Fraction-free diagonalization of rows $u_1, \ldots, u_r$ of B
if $r > 1$ **then**
  Choose $1 \le s < r$;
  for $x \in \{u_{s+1}, \ldots, u_r\}$ and $1 \le y \le m$ **do**
    Replace b_{xy} by $b_{x,y}^{[(u_1,v_1),\ldots,(u_s,v_s)]} / \left[a^{[(i_1,j_1),\ldots,(i_k,j_k)]}\right]^s$
  od;
  Diagonalize$(B, [(i_1, j_1), \ldots, (i_k, j_k)], ((u_1, v_1), \ldots, (u_s, v_s)))$;
  for $x \in \{u_1, \ldots, u_s\}$ and $1 \le y \le m$ **do**
    Replace b_{xy} by $b_{x,y}^{[(u_{s+1},v_{s+1}),\ldots,(u_r,v_r)]} / \left[a^{[(i_1,j_1),\ldots,(i_k,j_k)]\leftarrow[(u_1,v_1),\ldots,(u_s,v_s)]}\right]^{r-s}$
  od;
  Diagonalize$(B, [(i_1, j_1), \ldots, (i_k, j_k)] \leftarrow [(u_1, v_1), \ldots, (u_s, v_s)], ((u_{s+1}, v_{s+1}), \ldots, (u_r, v_r)))$
fi

So first rows $u_1, \ldots, u_s$ are used to clear columns $v_1, \ldots, v_s$ in rows $u_{s+1}, \ldots, u_r$. Then rows $u_1, \ldots, u_s$ are diagonalized (in columns $v_1, \ldots, v_s$). Next rows $u_{s+1}, \ldots, u_r$

are used to clear columns $v_{s+1}, \ldots, v_r$ in rows $u_1, \ldots, u_s$. Finally rows $u_{s+1}, \ldots, u_r$ are diagonalized (in columns $v_{s+1}, \ldots, v_r$).

Now we will prove that this algorithm is indeed fraction-free. First notice that when we first enter the loop the comment is true for $k = 0$. Now assume that at some stage in the algorithm the comment is true for $i_1, \ldots, i_k$ and $j_1, \ldots, j_k$ and that in the loop we choose $u_1, \ldots, u_r$ and $v_1, \ldots, v_r$. We will prove that then at the next entry of the loop we have

$$b_{xy} = a_{x,y}^{[(i_1,j_1),\ldots,(i_k,j_k)]\leftarrow[(u_1,v_1),\ldots,(u_r,v_r)]}. \tag{3}$$

This proves that the algorithm is fraction-free.

Now for $x \in \{1, \ldots, n\} \setminus \{u_1, \ldots, u_r\}$ (3) follows immediately from lemma 1. For $x \in \{u_1, \ldots, u_r\}$ we will prove (3) by induction on r. First notice that for $r = 1$ (then Diagonalize doesn't do anything) the statement follows from

$$\begin{aligned} a_{u_1,y}^{[(i_1,j_1),\ldots,(i_k,j_k)]} &= a^{[(i_1,j_1),\ldots,(i_k,j_k)]\leftarrow[(u_1,y)]} \\ &= a^{[(i_1,j_1),\ldots,(i_k,j_k)]\leftarrow[(u_1,y)]\leftarrow[(u_1,y)]} \\ &= a_{u_1,y}^{[(i_1,j_1),\ldots,(i_k,j_k)]\leftarrow[(u_1,v_1)]} \end{aligned}$$

Suppose now that the statement holds for $s < r$. Using lemma 1 we see that after the first loop in the algorithm Diagonalize we have for $x \in \{u_{s+1} \ldots, u_r\}$ and $1 \le y \le m$

$$b_{xy} = a_{x,y}^{[(i_1,j_1),\ldots,(i_k,j_k)]\leftarrow[(u_1,v_1),\ldots,(u_s,v_s)]}. \tag{4}$$

Using the induction hypothesis we see that after the first recursive call of Diagonalize we also have (4) for $x \in \{u_1, \ldots, u_s\}$ and $1 \le y \le m$. Using lemma 1 again we then see that after the second loop in the algorithm Diagonalize we have for $x \in \{u_1, \ldots, u_s\}$ and $1 \le y \le m$

$$b_{xy} = a_{x,y}^{[(i_1,j_1),\ldots,(i_k,j_k)]\leftarrow[(u_1,v_1),\ldots,(u_s,v_s)]\leftarrow[(u_{s+1},v_{s+1}),\ldots,(u_r,v_r)]}. \tag{5}$$

Finally, using the induction hypothesis again, we see that after the second recursive call of Diagonalize we also have (5) for $x \in \{u_{s+1}, \ldots, u_r\}$ and $1 \le y \le m$.

Since (5) equals $a_{x,y}^{[(i_1,j_1),\ldots,(i_k,j_k)]\leftarrow[(u_1,v_1),\ldots,(u_r,v_r)]}$ this proves the statement for $s = r$.

The values $a^{[(i_1,j_1),\ldots,(i_k,j_k)]}$ needed in the algorithm can be simply read as b_{i_k,j_k} and the values $a^{[(i_1,j_1),\ldots,(i_k,j_k)]\leftarrow[(u_1,v_1),\ldots,(u_s,v_s)]}$ as b_{u_s,v_s}.

Notice that when we always take $u_1, \ldots, u_r \notin \{i_1, \ldots, i_k\}$ we have the ordinary extended algorithm described in the introduction. If, in addition, we restrict our algorithm in such a way that only a triangular matrix is computed we have the ordinary unextended algorithm from the introduction. So we have proved in a uniform way the validity of all algorithms taken into consideration.

As in the ordinary fraction-free Gaussian elimination algorithms we can compute $b_{x,y}^{[(u_1,v_1),\ldots,(u_r,v_r)]}$ by expanding this determinant w.r.t. the column corresponding to y. Using lemma 1 we see that each minor used in this computation (which is independent of y) is divisible by $\left[a^{[(i_1,j_1),\ldots,(i_k,j_k)]}\right]^{r-1}$. One can use this to improve the algorithm in the usual way.

Also, as in the ordinary fraction-free Gaussian elimination algorithms, we have to be careful in our choice of the u's and v's. We must ensure that all divisions performed in the algorithm are allowed, i.e. no divisions by 0 must be made.

4. Linear programming

In this section we will show how the previous can be used in the simplex method, a method used to solve linear programming problems. For details on linear programming we refer the reader to Schrijver (1986).

Let the linear programming problem be given by $\max\{cx \mid x \geq 0; Ax \leq b \geq 0\}$ where A is an $m \times n$ matrix, $b \geq 0$ is a column vector of size m and c is a row vector of size n. All entries in A, b and c are supposed to be real numbers.

In the simplex method computations are performed in a so-called 'tableau'. A tableau is a matrix of the form

$$E = \begin{pmatrix} u & \delta \\ D & f \end{pmatrix},$$

where u is a row vector of size $n+m$, D is an $m \times (n+m)$ matrix and $f \geq 0$ is a column vector of size m. One starts with the tableau where $D = (A \mid I)$, $u = (-c \mid 0)$, $f = b$ and $\delta = 0$.

Now one tries to diagonalize the matrix D (subject to some more constraints), but one pivots the whole matrix E. The simplex method is a 1-step elimination method.

In the simplex method one repeatedly does the following:

> Choose j such that $u_j < 0$ and choose l such that $d_{lj} > 0$ and $f_l/d_{lj} = \min\{f_s/d_{sj} \mid d_{sj} > 0\}$. Use d_{lj} to perform the next pivoting. Notice that there are still several ways of choosing j and l.

If in the above a suitable j does not exist, one has found an optimal solution. If a suitable l does not exist, the maximum is not bounded.

In the simplex method pivots d_{ij} are used where the ith row and jth column could already have contributed a pivot. So this is an example of random Gaussian elimination.

In practice (for very big problems) the simplex method is mostly performed numerically. As is commonly known, numerical computations can lead to incorrect results, as the following example also shows.

Example In this example we will solve the linear programming problem where

$$A = \begin{pmatrix} 111 & 221 \\ 110 & 222 \end{pmatrix}, b = \begin{pmatrix} 100 \\ 100 \end{pmatrix} \text{ and } c = \begin{pmatrix} 113 & 225 \end{pmatrix}.$$

We will perform both the numerical and the exact fractional simplex method. The pivots used are marked by a box. In the numerical computation we compute with an accuracy of three digits.

$$\begin{pmatrix} -113. & -225. & 0 & 0 & 0 \\ \boxed{111.} & 221. & 1. & 0 & 100. \\ 110. & 222. & 0 & 1. & 100. \end{pmatrix} \to \begin{pmatrix} 0 & 0 & 1.02 & 0 & 102. \\ 1.00 & 1.99 & .00901 & 0 & .901 \\ 0 & 3. & -.991 & 1. & .9 \end{pmatrix}$$

We can now read in the fifth column that the optimal solution is $(.901, 0)$ and its corresponding optimal value is 102.

$$\begin{pmatrix} -113 & -225 & 0 & 0 & 0 \\ \boxed{111} & 221 & 1 & 0 & 100 \\ 110 & 222 & 0 & 1 & 100 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & \frac{-2}{111} & \frac{113}{111} & 0 & \frac{11300}{111} \\ 1 & \frac{221}{111} & \frac{1}{111} & 0 & \frac{100}{111} \\ 0 & \boxed{\frac{332}{111}} & \frac{-110}{111} & 1 & \frac{100}{111} \end{pmatrix} \rightarrow$$

$$\rightarrow \begin{pmatrix} 0 & 0 & \frac{84}{83} & \frac{1}{166} & \frac{8450}{83} \\ 1 & 0 & \frac{111}{166} & \frac{-221}{332} & \frac{25}{83} \\ 0 & 1 & \frac{-55}{166} & \frac{111}{332} & \frac{25}{83} \end{pmatrix}$$

In the fifth column we can read that the optimal solution is $(\frac{25}{83}, \frac{25}{83})$ and that the corresponding optimal value is $\frac{8450}{83}$. Transformed to decimal numbers this gives $(.301, .301)$ and 102. We see that the numerical solution is not even close to the correct solution. Notice that the numerical optimal value is correct in this example.

As is commonly known the disadvantage of exact fractional computations are the numerous expensive gcd computations needed in order to represent fractions by expresssions of manageable size. By using the algorithm as described in the previous section we obtain a fraction-free version of the simplex method, avoiding the gcd computations.

In the fraction-free method, if we choose l and j for the pivot, then the pivot actually is $a_{l,j}^{[(i_1,j_1),\dots,(i_k,j_k)]} = a^{[(i_1,j_1),\dots,(i_k,j_k)]\leftarrow[(l,j)]}$, which equals the factor by which we divide in the next step. So the factor by which we divide in the algorithm is always the previous pivot. Since the pivots are always positive, this ensures that the sign of an entry does not change when we divide by those factors. This is important to notice since this ensures that $\leq$-relations keep valid and thus the fraction-free simplex method is correct. The problem of negative divisors is addressed in the next section.

Since during the algorithm each entry of E is of the form $e_{i,j}^{[(i_1,j_1),\dots,(i_k,j_k)]}$ we can get bounds for these entries, for example by using Hadamard's bound for determinants (see Horn and Johnson (1985)).

Example In this example we will solve the linear programming problem where

$$A = \begin{pmatrix} 12 & 9 & 2 \\ 3 & 5 & 17 \end{pmatrix}, b = \begin{pmatrix} 1 \\ 7 \end{pmatrix} \text{ and } c = \begin{pmatrix} 2 & 4 & 13 \end{pmatrix}.$$

We will perform both the fractional and the fraction-free simplex method. The pivots used are marked by a box.

$$\begin{pmatrix} -2 & -4 & -13 & 0 & 0 & 0 \\ \boxed{12} & 9 & 2 & 1 & 0 & 1 \\ 3 & 5 & 17 & 0 & 1 & 7 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & \frac{-5}{2} & \frac{-38}{3} & \frac{1}{6} & 0 & \frac{1}{6} \\ 1 & \boxed{\frac{3}{4}} & \frac{1}{6} & \frac{1}{12} & 0 & \frac{1}{12} \\ 0 & \frac{11}{4} & \frac{33}{2} & \frac{-1}{4} & 1 & \frac{27}{4} \end{pmatrix} \rightarrow$$

$$\rightarrow \begin{pmatrix} \frac{10}{3} & 0 & \frac{-109}{9} & \frac{4}{9} & 0 & \frac{4}{9} \\ \frac{4}{3} & 1 & \frac{2}{9} & \frac{1}{9} & 0 & \frac{1}{9} \\ \frac{-11}{3} & 0 & \boxed{\frac{143}{9}} & \frac{-5}{9} & 1 & \frac{58}{9} \end{pmatrix} \rightarrow \begin{pmatrix} \frac{7}{13} & 0 & 0 & \frac{3}{143} & \frac{109}{143} & \frac{766}{143} \\ \frac{18}{13} & 1 & 0 & \frac{17}{143} & \frac{-2}{143} & \frac{3}{143} \\ \frac{-3}{13} & 0 & 1 & \frac{-5}{143} & \frac{9}{143} & \frac{58}{143} \end{pmatrix}$$

We can now read in the sixth column that the optimal solution is $(0, \frac{3}{143}, \frac{58}{143})$ and its corresponding optimal value is $\frac{766}{143}$.

$$\begin{pmatrix} -2 & -4 & -13 & 0 & 0 & 0 \\ \boxed{12} & 9 & 2 & 1 & 0 & 1 \\ 3 & 5 & 17 & 0 & 1 & 7 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & -30 & -152 & 2 & 0 & 2 \\ 12 & \boxed{9} & 2 & 1 & 0 & 1 \\ 0 & 33 & 198 & -3 & 12 & 81 \end{pmatrix} \rightarrow$$

$$\rightarrow \begin{pmatrix} 30 & 0 & -109 & 4 & 0 & 4 \\ 12 & 9 & 2 & 1 & 0 & 1 \\ -33 & 0 & \boxed{143} & -5 & 9 & 58 \end{pmatrix} \rightarrow \begin{pmatrix} 77 & 0 & 0 & 3 & 109 & 766 \\ 198 & 143 & 0 & 17 & -2 & 3 \\ -33 & 0 & 143 & -5 & 9 & 58 \end{pmatrix}$$

In the sixth column we read the needed values as in the previous computation. Notice that we have to divide these values by 143, the last pivot used.

A special case in linear programming is when E is in fact an integer matrix and A is a totally unimodular matrix, i.e. each minor of A is either $-1, 0$ or 1. One can show that in that case a solution to the linear programming problem has also integer components. From now on assume that we are in this totally unimodular case. Then the matrix $(A \mid I)$ is also totally unimodular. Again using the fact that during the algorithm each entry of E is of the form $e_{i,j}^{[(i_1,j_1),\ldots,(i_k,j_k)]}$, we see that the entries of D will be $-1, 0$ or 1 during the algorithm (this was also proved in Bronstein, Mulders and Weil (1997)). Since the pivots chosen are always positive it follows that in this case all pivots are equal to 1.

Since the factor by which we divide in the algorithm is always the previous pivot, this is also 1. We see that in this case the fraction-free method coincides with the fractional method. One can use this to optimize the algorithm by avoiding unnecessary multiplications and divisions by 1.

In this case the bounds for the entries during the algorithm are especially simple. The entries in the u-vector equal the determinant of a matrix

$$\begin{pmatrix} \tilde{A} \\ \tilde{c} \end{pmatrix},$$

where $\tilde{A}$ (resp. $\tilde{c}$) is a submatrix of $(A \mid I)$ (resp. subvector of $(-c \mid 0)$). Since $(A \mid I)$ is totally unimodular we see by expanding this determinant to the last row that this is $\leq \min(m+1, n)C$ where C is a bound for the entries in c. For the entries in f we get the determinant of a matrix

$$\begin{pmatrix} \tilde{A} & \tilde{b} \end{pmatrix},$$

where $\tilde{A}$ (resp. $\tilde{b}$) is a submatrix of $(A \mid I)$ (resp. subvector of b). By expanding this to the last column we see that this is $\leq mB$ where B is a bound for the entries in b. In the same way we get a bound of $\min(m,n)mBC$ for the entry δ. These bounds can be used when implementing the simplex algorithm. For example, if one can use the bounds to show that during the algorithm all entries can be represented by single-precision integers, one can use this fact in order to optimize the code for the algorithm. This has been done in an implementation of an algorithm described in Bronstein, Mulders and Weil (1997).

One gets a two-step simplex algorithm by repeatedly doing the following:

> Choose j and l as in the 1-step method. Instead of performing the pivoting with d_{lj} on the whole matrix E, one now only computes those entries in the pivoted matrix which are needed to choose j and l in the next step. In particular, the new entries in the u-row have to be computed to choose the new j. Then the new entries in the f-column and jth column of D have to be computed to choose the new l. When one has chosen in this way two succesive values for j and l one can perform the two-step algorithm on the whole matrix.

Taking the fraction-free version of the above, we get a two-step fraction-free simplex algorithm and in the same way one can design multi-step algorithms.

We've implemented the fractional, 1-step fraction-free and 2-step fraction-free simplex method in Maple V.4, using Bland's pivoting rule (see Schrijver (1986)). The following table shows some timings (in CPU seconds) on a DEC Alpha 500/333, indicating the improvement by using fraction-free methods. The matrices and vectors used have random, s digits long, integer entries.

m	n	s	fractional	1-step fraction-free	2-step fraction-free
40	35	2	47	21	19
50	60	2	259	108	82
40	30	10	239	63	45
40	40	10	959	207	147

5. Solving $Ax = b$ $(x \geq 0)$

In this section we will show how the previous can be used to find a solution of $Ax = b$ $(x \geq 0)$, where A is an $m \times n$ matrix and b is a column vector of size m. All entries in A and b are supposed to be real numbers.

Now we will work in a tableau of the form

$$E = \begin{pmatrix} D & f \end{pmatrix},$$

where D is an $m \times n$ matrix and f is a column vector of size m. We start with the tableau where $D = A$ and $f = b$. First we perform ordinary Gaussian elimination on E to transform D into a matrix having the $m \times m$ identity matrix as a submatrix (for simplicity we assume that A has rank m). Say the $\sigma(i)$th column of D is the ith standard basis vector. Then the f-column will give a (not necessarily positive) linear dependence of b on m linearly independent columns of A, i.e. $b = \sum f_i A_{\sigma(i)}$. Now we apply the algorithm indicated by the proof of the fundamental theorem of linear programming (see Schrijver (1986)), i.e. repeatedly do the following:

> Choose l such that $f_l < 0$ and $\sigma(l)$ is minimal. Choose j minimal such that $d_{lj} < 0$. Use d_{lj} to perform the next pivoting.

When l does not exist we have found a solution, when j does not exits there is no solution.

All that is said in the previous section applies to this application. In particular we have a fraction-free version of this algorithm and even a multi-step version can be designed.

Notice that in this case the pivots used in the second step of the algorithm are negative. In the fraction-free algorithms we have to divide the possible solution, which we can read in the f-column, by the last pivot used. When this last pivot is negative, division by it will change signs, so we have to adjust the selection of the next pivot (replace $<$ by $>$) according to the sign of the last pivot.

Example In this example we will compute a solution of $Ax = b$ $(x \geq 0)$ where

$$A = \begin{pmatrix} 7 & -1 & -2 \\ 5 & 3 & -6 \end{pmatrix} \text{ and } b = \begin{pmatrix} 0 \\ -7 \end{pmatrix}.$$

We perform both the ordinary and fraction-free algorithm. Pivots are marked by a box.

$$\begin{pmatrix} \boxed{7} & -1 & -2 & 0 \\ 5 & 3 & -6 & -7 \end{pmatrix} \to \begin{pmatrix} 1 & \frac{-1}{7} & \frac{-2}{7} & 0 \\ 0 & \boxed{\frac{26}{7}} & \frac{-32}{7} & -7 \end{pmatrix} \to \begin{pmatrix} 1 & 0 & \boxed{\frac{-6}{13}} & \frac{-7}{26} \\ 0 & 1 & \frac{-16}{13} & \frac{-49}{26} \end{pmatrix} \to$$

$$\to \begin{pmatrix} \frac{-13}{6} & 0 & 1 & \frac{7}{12} \\ \boxed{\frac{-8}{3}} & 1 & 0 & \frac{-7}{6} \end{pmatrix} \to \begin{pmatrix} 0 & \frac{-13}{16} & 1 & \frac{49}{32} \\ 1 & \frac{-3}{8} & 0 & \frac{7}{16} \end{pmatrix}$$

We can now read in the fourth column the solution $(\frac{7}{16}, 0, \frac{49}{32})$.

$$\begin{pmatrix} \boxed{7} & -1 & -2 & 0 \\ 5 & 3 & -6 & -7 \end{pmatrix} \to \begin{pmatrix} 7 & -1 & -2 & 0 \\ 0 & \boxed{26} & -32 & -49 \end{pmatrix} \to \begin{pmatrix} 26 & 0 & \boxed{-12} & -7 \\ 0 & 26 & -32 & -49 \end{pmatrix} \to$$

$$\to \begin{pmatrix} 26 & 0 & -12 & -7 \\ \boxed{32} & -12 & 0 & 14 \end{pmatrix} \to \begin{pmatrix} 0 & -26 & 32 & 49 \\ 32 & -12 & 0 & 14 \end{pmatrix}$$

As in the previous computation we can read the needed values in the fourth column. Notice that we have to divide these values by 32, the last pivot used.

We've implemented a fractional, 1-step fraction-free and 2-step fraction-free version of the above algorithm. The following table shows some timings (in CPU seconds) on a DEC Alpha 500/333, indicating the improvement by using fraction-free methods. The matrices used have random, s digits long, integer entries. The vectors are built as Ax where x is vector having random, s digits long, positive integer entries.

m	n	s	fractional	1-step fraction-free	2-step fraction-free
40	60	2	118	30	26
60	90	2	1208	314	244
30	45	10	296	37	29
40	60	10	665	230	106

One can also solve the second step of the previous algorithm by introducing a new variable y and solving $\max\{-y \mid y \geq 0; x \geq 0; Bz = b\}$ where B is the matrix A concatenated with an all-one column vector and z is the vector x concatenated with y. This last problem can be solved by the method from the previous section. In this way the problem is solved in the simplex package of Maple.

In general the last method is faster then the first method. This is due to the fact that the number of pivots used in the second method is less than in the first one.

We've also implemented a fractional, 1-step fraction-free and 2-step fraction-free version of the second method. The timings on the same examples as above are listed in the following table.

m	n	s	fractional	1-step fraction-free	2-step fraction-free
40	60	2	75	31	21
60	90	2	621	161	118
30	45	10	164	37	25
40	60	10	516	115	73

6. Conclusion

The generalized Sylvester identity is a generalization of the well-known Sylvester identity, relating a hyperdeterminant of a matrix to the determinant of that matrix. Using this generalized identity one can design fraction-free versions of all kinds of algorithms, not only involving ordinary Gaussian elimination but also random Gaussian elimination.

A class of algorithms involving random Gaussian elimination is provided by linear programming and its related topics. Using fraction-free versions of those algorithms we get more efficient algorithms for solving linear programming problems exactly. Notice that it is hard, if not impossible, to use modular methods in these kinds of algorithms, since relational properties ($\leq, \geq$) among intermediate results are needed during the algorithms.

References

Bareiss, E.H. (1968). Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination. *Math. Comp.* **22**(103), 565–578.

Bareiss, E.H. (1972). Computational Solutions of Matrix Problems Over an Integral Domain. *J. Inst. Maths Applcs* **10**, 68–104.

Bronstein, M., Mulders, T., Weil, J.-A. (1997). On Symmetric Powers of Differential Operators. *Submitted to ISSAC'97.*

Horn, R.A., Johnson, C.R. (1985). *Matrix Analysis*. Cambridge University Press.

Schrijver, A. (1986). *Theory of linear and integer programming*. Wiley.